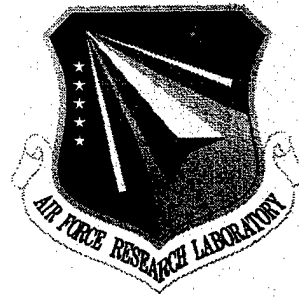


AFRL-IF-RS-TR-2000-4
Final Technical Report
January 2000



FORMAL ALTERNATIVES MANAGEMENT INTEGRATING LOGICAL INFERENCE AND RATIONALE (FAMILIAR)

Knowledge Evolution, Inc.

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. D899

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

DTIC QUALITY INSPECTED 4

20000224 098

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

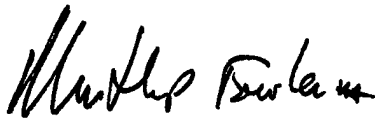
AFRL-IF-RS-TR-2000-4 has been reviewed and is approved for publication.

APPROVED:



DEBORAH A. CERINO
Project Engineer

FOR THE DIRECTOR:



NORTHROP FOWLER
Technical Advisor
Information Technology Division

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

FORMAL ALTERNATIVES MANAGEMENT INTEGRATING LOGICAL
INFERENCE AND RATIONALE (FAMILIAR)

Sidney C. Bailin and Dean T. Allemang

Contractor: Knowledge Evolution, Inc.

Contract Number: F30602-96-C-0284

Effective Date of Contract: 27 August 1996

Contract Expiration Date: 30 August 1999

Short Title of Work: Formal Alternatives Management
Integrating Logical Inference and
Rationale (Familiar)

Period of Work Covered: Aug 96 – Aug 99

Principal Investigator: Sidney C. Bailin

Phone: (202) 467-9588

AFRL Project Engineer: Deborah A. Cerino

Phone: (315) 330-1445

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

This research was supported by the Defense Advanced Research
Projects Agency of the Department of Defense and was monitored
by Deborah A. Cerino, AFRL/IFTD, 525 Brooks Road, Rome, NY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE JANUARY 2000		3. REPORT TYPE AND DATES COVERED Final Aug 96 - Dec 99
4. TITLE AND SUBTITLE FORMAL ALTERNATIVES MANAGEMENT INTEGRATING LOGICAL INFERENCE AND RATIONALE (FAMILIAR)			5. FUNDING NUMBERS C - F30602-96-C-0284 PE - 62301E PR - D899 TA - 01 WU - 01	
6. AUTHOR(S) Sidney C. Bailin and Dean T. Allemang				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Prime: Knowledge Evolution, Inc. 1050 17th Street, NW, Suite 520 Washington DC 20036 Sub: Synquiry Technologies, Ltd. 1 Williston Road Suite 4 Belmont MA 02178			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington Virginia 22203-1714 Air Force Research Laboratory/IFTD 525 Brooks Road Rome NY 13441-4505			10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2000-4	
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Deborah A. Cerino/IFTD/(315) 330-1445				
12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The FAMILIAR project goals were to: 1) position software design rationale capture in the context of alternatives management, i.e., systematic exploration of design alternatives and their trade-offs against the goals of the system under development; and 2) formalize the capture and presentation of design rationale through Functional Representation. The FAMILIAR technology is a union of alternatives management and Functional Representation. It is based on earlier technologies. KAPTUR and ZD 1, which explored these respective approaches. FAMILIAR built on lessons learned from those systems and provided a more robust tool that supports a range of rationale capture situations from completely informal to fully formal. FAMILIAR also supports the evolution towards increasing formalism as a domain matures.				
14. SUBJECT TERMS Rationale Capture, Design Record, Decision Support, Formal Specification, Automated Analysis, Domain Modeling			15. NUMBER OF PAGES 44	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Executive Summary

This document summarizes the work performed under the FAMILIAR project within DARPA's Evolutionary Design of Complex Software (EDCS) program. FAMILIAR was part of the Rationale Capture (RC) thrust of EDCS. Rationale capture is a way of mitigating risk in critical design decisions. Risk in such situations derives from the multi-layered, multi-dimensional nature of the systems being designed, and from uncertainty about changing conditions in which the systems will be deployed. RC encourages the rigorous evaluation of alternatives before committing to a design choice. It enables designers to revisit and re-evaluate decisions in the event of changed circumstances or new information.

FAMILIAR is a decision support tool that encourages systematic exploration of design alternatives and their tradeoffs. FAMILIAR applies this idea to the evolution of a target system. Successive versions of the target system represent "snapshots" of an ongoing decision process, which can be compared and contrasted through the rationales behind the decisions.

FAMILIAR represents design knowledge in terms of goals, alternatives, features, and components. *Goals* are the objectives of the current design task. *Alternatives* are different ways of achieving the goals. Alternatives are arranged in a tree where each branch consists of successively more detail about potential solutions. *Features* are the dimensions along which the alternatives differ. Features provide a way to compare and contrast specific aspects of alternative designs and to perform "what if" analyses by changing some aspects while keeping others constant. Each alternative may be composed of several *components* (e.g., subsystems or low-level components). FAMILIAR maintains the linkage between choices at the component level and their impact on higher-level system goals.

FAMILIAR uses a technique called Functional Representation (FR) to formalize the expression of design rationale. This allows FAMILIAR to analyze design decisions and provide feedback to the designer. The role of FR in FAMILIAR is to enable reasoning about *functional rationale*, i.e., understanding the role that a particular function plays in realizing the goals of the target system. FR allows a designer to discover what happens if one or more of the goals change, or if one function is replaced by another, or if a function is removed or added. Unlike most formal methods, FR allows the formalization of selected aspects of a complex system without requiring a complete formal representation. This makes it much more feasible for real-world use.

We have implemented several mechanisms for capturing design rationale non-intrusively, e.g., from e-mail messages, placing the information in the FAMILIAR database, and incrementally formalizing it as designers' understanding of the target system evolves. Together, these capabilities provide a way of managing uncertainty, mitigating risk, and building on best practice in system development.

Table of Contents

EXECUTIVE SUMMARY	I
TABLE OF CONTENTS	II
1 INTRODUCTION	1
2. OBJECTIVES	2
3. ACCOMPLISHMENTS	3
3.1 FORMAL ALTERNATIVES MANAGER (FAM)	4
3.2 ENHANCEMENTS TO THE ZD FUNCTIONAL REPRESENTATION SYSTEM	5
3.3 INTEGRATED RATIONALE CAPTURE AND REPRESENTATION	7
3.4 ACCOMPLISHMENTS VS. OBJECTIVES	8
3.5 ACCOMPLISHMENTS VS. EVALUATION CRITERIA	9
4. APPROACH	9
4.1 YEAR ONE: CON-OPS AND INITIAL PROTOTYPE	10
4.2 YEAR TWO: DESIGN RATIONALIZATION AND USABILITY ENHANCEMENTS	11
4.3 YEAR THREE: IMPROVED CON-OPS AND CLIENT-SERVER ARCHITECTURE	13
5. LESSONS LEARNED	14
5.1 NON-INTRUSIVE RATIONALE CAPTURE	14
5.2 INTEGRATED DISPLAY	15
5.3 PRO-ACTIVE GUIDANCE TO THE USER	16
5.3.1 <i>Pros and Cons of a Process-Based Interface</i>	17
5.4 NEED FOR DBMS FUNCTIONALITY	17
5.5 FAMILIAR AS CORPORATE MEMORY	18
5.6 FAM AS AN INTELLIGENT PORTAL	19
5.7 DESIGN TRACE VS. DOMAIN MODEL	19
5.8 RELATIONSHIP WITH WINWIN	20
5.9 NEED FOR RULES	21
5.10 FEATURE COMBINATIONS	21
5.11 TRADEOFF CALCULATIONS	22
5.12 NEED FOR A LIGHTWEIGHT USER INTERFACE	23
5.13 JAVA FRAGILITY	23
5.14 IMPACT OF USING A THEOREM PROVER IN ZD	24
5.15 INTEROPERABILITY OF FORMAL LANGUAGES	25
6. FUTURE DIRECTIONS	25
LIST OF ACRONYMS	28

1 INTRODUCTION

This document summarizes the work performed under a project entitled Formal Alternatives Management Integrating Logical Inference and Rationales (FAMILIAR). FAMILIAR was part of the Evolutionary Design of Complex Systems (EDCS) program, sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL).

FAMILIAR was part of the Rationale Capture (RC) thrust of EDCS. The observation motivating RC was that the evolution of complex systems involves a web of inter-related design decisions with many dimensions of choice. There is often significant uncertainty about the potential impact of the decisions and the changing conditions in which they will be deployed. In order to minimize the risk in such a process and to optimize the decisions made, it is necessary to articulate the rationales for decisions so that they can be rigorously evaluated and, if necessary, revisited in the future.

FAMILIAR sought to situate RC in the context of *alternatives management*, that is, the systematic exploration of alternative design decisions and their tradeoffs against the goals of the system under development (the *target system*). In this respect, FAMILIAR represents a marriage of RC with certain ideas from the discipline of domain modeling. In domain modeling, alternative ways of designing a particular type of system are described. It is understood that in certain contexts, certain design approaches will be more appropriate than others. The domain model, together with its associated database of reusable components (or *assets*), defines the space of available solutions and indicates the conditions under which either one or another solution is to be preferred.

FAMILIAR adapts this idea to the evolution of a single system (or, perhaps, different versions of a single system). This adaptation amounts to broadening the definition of *domain* so that it can refer to the space of possible designs for a single system, or for multiple versions of a system, or for similar but distinct systems. By broadening the notion of domain in this way, FAMILIAR establishes itself as both a design tool and a domain modeling tool, with emphasis in both respects on alternatives management and rationale capture.

The other primary innovation of FAMILIAR was to formalize the capture and presentation of design rationale. Formalization is achieved through *functional representation* (FR), which is a method of expressing the teleology of various parts, or facets, of a system. FR has several advantages over other formal approaches to system specification and design. In FAMILIAR, there are two principal benefits of FR:

1. It allows formalization of selected aspects of a complex system without requiring a complete formal representation.

2. It decouples teleology from structure. A designer can reason about a particular functional aspect of the target system whether or not it corresponds to a defined software or hardware component.

The role of FR in FAMILIAR is to enable reasoning about *functional rationale*, i.e., understanding the role that a particular function plays in realizing the goals of the target system. FR allows a designer to discover what happens if one or more of the goals change, or if one function is replaced by another, or if a function is removed or added.

FAMILIAR, as its name implies, is a union of alternatives management and functional representation. The project took as its starting point an existing tool in each of these two areas. For alternatives management, the starting point for FAMILIAR was the KAPTUR (pronounced "capture") tool, which originated as a domain modeling tool that emphasized rationale capture for decision support. For functional representation, the springboard was the ZD (pronounced "zeddy") tool, which applies FR to software design. These two tools were the legacy that the two PIs brought with them to FAMILIAR: Dr. Bailin was the originator of KAPTUR, and Dr. Allemang's research was the basis for the ZD tool. Lessons were learned through the earlier development of both tools. It was the intention of the PIs to apply these lessons to an integrated FAMILIAR tool that would improve upon both of its precedents, at the same time enhancing each through their complementary capabilities.

2. OBJECTIVES

In the FAMILIAR proposal, the following benefits were identified as goals of the project:

- Non-intrusive and cost-effective design rationale capture

Despite the recognized importance of design rationale, and the prevalence of reasoning about alternatives in the software design process, it is the exception rather than the rule when rationales are recorded for later consumption. The main reason for this is the effort that must be expended to record design rationale, and the perceived interference of this effort with the design process itself.

One of the goals for FAMILIAR was to minimize the effort involved in capturing design rationale, and to weave rationale capture as seamlessly as possible into the design process.

- Enhanced automated support for software/system understanding

We envisioned a tool that could automatically infer and provide explanations of certain software design properties through FR.

- Enhanced ability to anticipate the impact of intended changes to a system

Accurately predicting the effects of changes has proven to be one of the most difficult aspects of evolving a complex system. Through automated reasoning using FR, FAMILIAR was intended to identify potential problems with a proposed change. For

example, by tracing the fulfillment of *provisos* (conditions that must be met for declared functional goals to be achieved), FR could identify gaps in a design. The range of changes susceptible to such analysis would depend on the extent to which a system has been modeled using FR.

- Support for increasing formalization as a domain matures

In KAPTUR-style alternatives management, design alternatives are characterized by features selected from a domain ontology. By combining this with the more formal FR approach, which can be applied to all or part of a system design, we intended to support a spectrum of rationale representations, ranging from completely informal to completely formal. As a domain matures, FAMILIAR would support increasing levels of formalization by supporting increasingly rich semantics for the features, culminating in a functional representation where appropriate.

- Traceability of evolving design decisions

FAMILIAR was intended to support a trail of "snapshots" of points visited in the space of design alternatives as a system evolves. By borrowing the idea of a *family of systems* from domain modeling, we would be able to represent these multiple snapshots in a way that facilitated comparison and understanding the differences. This amounts to understanding the evolution of the target system.

- Decision support through disciplined evaluation of design alternatives

Ultimately, the motivation for rationale capture is to improve the quality of design decisions. During initial design, articulating rationales can help clarify the desirability of decisions. During system evolution, previously recorded rationales provide a basis for assessing proposed changes. Rationales, however, are only as useful as the underlying reasoning is rigorous. A rationale that is not based on sound evaluation of alternatives is of minimal use, except perhaps as a negative example. One of the goals of FAMILIAR was to emphasize the use of recorded rationale for decision support, and this entails supporting the systematic evaluation of design alternatives.

- Increased integration and feedback between lifecycle phases

Because FR can be used at various levels of abstraction, it can serve as a link between abstract functional design and the more concrete activity of configuring components. We expected this to facilitate backtracking to the earlier design phases when necessary, enabling designers to move back and forth between abstract specification and concrete configuration more fluidly.

3. ACCOMPLISHMENTS

Our main accomplishment is the development of an integrated rationale capture system combining alternatives management with formal reasoning. The Formal Alternatives Manager (FAM) acts as an intelligent portal to the knowledge needed for decision

support during software (or other) design processes. The enhanced ZD tool supports the evaluation of designs and helps refine proposed solutions.

We demonstrated the integrated system's use to support a tradeoff- and rationale-based design process for both software and logistics plans. The demonstrations included automated reasoning to detect and recommend fixes for missing functions.

The following sub-sections provide more detail about technical accomplishments within the alternatives manager and the FR system, as well as about their integration with each other and with other EDCS technologies.

3.1 Formal Alternatives Manager (FAM)

FAM represents a re-thinking of the ideas that went into the earlier KAPTUR tool. The advances concern usability as well as richer semantics for the data structures involved in alternatives management.

FAM models evolving design knowledge as a collection of interrelated hierarchies: goals, alternatives, features, and components. *Goals* represent the objectives of the design task currently being performed. Each goal has a numerical importance assigned to it relative to other goals. This information allows FAM to provide decision support to the user in choosing among *alternatives*. The alternatives hierarchy is a categorization of designs. Moving down the hierarchy leads to more specialized knowledge. *Features* are the dimensions along which alternatives differ from each other. Because any complete design is a combination of many aspects, the feature hierarchy provides a way to compare and contrast specific aspects of alternative designs and to perform “what if” analyses by changing some aspects while keeping others constant.

Each alternative design may be composed of several connected *components* (e.g., subsystems or low-level components). Each component comes from its own domain, where it is probably just one out of several alternatives in that domain. The component's distinguishing features have an impact on the features of the containing design. FAM maintains these interconnections, propagating changes so that the hierarchies remain mutually consistent, providing the user with a simple means of viewing the knowledge interconnections relevant to a given design task.

While FAM provides a front-end for organizing a team's knowledge, the knowledge itself — specific goals, designs, or feature documentation — is maintained on the Web external to FAM and accessed through an easy point-and-click interface. Because this information may be output from or input to a diverse set of tools, or may be stored in diverse databases, FAM provides a number of methods for automated, non-intrusive rationale capture.

One automated capture mechanism is the ability to import text files containing FAM-decipherable information. We view this as a general purpose mechanism for loosely

interfacing with discipline-specific design evaluation tools (e.g., cost models, performance analyzers, etc.). In addition, text import can be used a fast way to create a large database, where item-by-item input into the tool would be too time consuming. FAM also has a published Java-API, so tighter tool integration can be achieved, if desired.

An e-mail capture capability provides a way to annotate FAM items (design alternatives, features, goals, tradeoffs) with commentary that occurs naturally in the e-mail dialogues between team members. E-mail that is cc'd to a FAM e-mail server is automatically stored and indexed at a designated URL. When the sender next brings up FAM, she is prompted (optionally) to assign portions of the message to one or more FAM items, using a simple highlight-and-click interface.

We also implemented a rule engine that can be used to extend the semantics of alternatives, features, goals, components, and tradeoffs, either generically or for a specific project or domain. The rule engine is an adaptation of the Hendrix tool, which was developed under AFRL's Knowledge-Based Software Assistant Project.

3.2 Enhancements to the ZD Functional Representation System

In tailoring the ZD tool to support functional rationale capture and inter-operate with FAM, we implemented the following specific enhancements to existing capabilities:

- Integration of functional representation with an automatic theorem prover
The theorem prover that was chosen for this was the NQTHM of Boyer and Moore,
- Compatibility Constraint Satisfaction Problem (CCSP) algorithm
This was one of the main configuration capabilities of ZD1. We implemented it in the formalism of the theorem prover.
- Proviso Proving and Propagation (PPP) algorithm
The algorithm is derived from Allemang's dissertation. We implemented it using the formalism of the theorem prover.

Further accomplishments reflected new capabilities for ZD. Some of these were inspired by other EDCS work:

- Extension of the Rapide analysis of the X-Open 2-phase commit database reference architecture

ZD represents a system in terms of states, changes from one state to another, and the agents that effect these changes. Because ZD allows this information to be represented formally, it is possible to reason about the circumstances in which an agent will bring about a desired state.

At one point in the 2-phase commit strategy, the agent that will effect the state change is chosen. The choice is mediated by certain constraints on when agents may act. This raises the question: Is the set of agents given in the reference architecture, acting under these constraints, sufficient to ensure that the state change will occur as planned under all circumstances?

ZD was extended with a capability to prove when this is the case. Specifically, ZD proves that the rollback and commit actions, when used according to the 2-phase commit strategy, guarantee the continued consistency of the distributed database.

We also extended the example to accommodate some credible changes in the assumptions made by the X-Open reference architecture. For example, instead of assuming that each database will return either "yes" or "no" when queried "will you commit if asked?" we broadened the assumption to include a third possible result of *time out without reply*. ZD was able to prove that the original reference architecture would fail to ensure consistency in this case, while a slight modification of the architecture can be proven to ensure consistency. These results complement those obtained by Rapide for the same example.

- Extension of ZD to accommodate knowledge sources

At Georgia Tech, the development of SIRRINE (and before that, Autognostic) concentrated on the nature of *knowledge sources* in software systems. The ZD architecture was extended to reflect this capability by the addition of *domains*, which are shared sources of formal information. Domains can be used to reason about the correctness of connections spanning disparate parts of the analyzed system's architecture.

- A simple form of a survivability determination

This demonstration uses a functional representation to determine possible alternative plans. The formal representation, along with other functional representation capabilities (PPP and CCSP), allow the system to provide detailed information about the feasibility of each alternative.

Survivability analysis is a natural application of a functional representation like ZD, since ZD represents not only what services are supplied by a component, but also what services were required by that component in its context. This means that when a component fails, the suitability of another component, which provides a different service, can nevertheless be evaluated. At the EDCS demonstration in 1999, we demonstrated a simple form of survivability analysis based on this principle. Because of the formal system in ZD (supported by a theorem prover), and the name-space bookkeeping done by ZD, it was possible to develop this capability in about a half-day.

Other accomplishments include documentation of the use of ZD, including several examples involving Y2K issues. Details of the ZD system that implements these

capabilities can be found in the ZD tutorial, available at the functional representation for software web page (www.synquiry.com).

3.3 Integrated Rationale Capture and Representation

Much of the effort expended in the FAMILIAR project was committed to integrating the system. One of the major goals of the project was to combine the alternatives management capabilities of FAM with the functional representation capabilities of ZD. Our strategy for this integration was to make use of as much EDCS technology as possible. In particular, we demonstrated the following:

- **Integration of FAMILIAR with ACME**

In its first year, FAMILIAR was in two parts, an alternatives management system and a functional representation system. A particular challenge faced in the project was how to share information about the decomposition of a system between these two components. Since the information that was to be shared concerned the architecture of the system, we decided to use the ACME architecture interchange language to mediate the connection.

We provided a capability for FAM to express and export decompositions of alternative designs in ACME. ZD was given the capability to import these as functional representations. Since ZD can make use of more information than simple decomposition, we integrated the system with the ACMESTudio tool. This allows a user to augment the decomposition description in ACME.

Finally, we provided a means for ZD to report its findings as annotations in ACME. This again can be displayed by ACMESTudio. An example of the capability was demonstrated at the second EDCS Demo Days. The example now appears on the functional representation for software web page (www.synquiry.com).

- **Distributing the application**

The various parts of the integrated application described above have different requirements for the platforms on which they run. The most restrictive of these is ACMESTudio, which runs only on Windows (tm) platforms. A major challenge in engineering the system was to get all the pieces, running in different languages, to inter-operate. We took advantage of another EDCS technology, Franz Orblink, to mediate the inter-operation. ZD, which is written in Lisp, communicated with ACME through Orblink. ACME, which was available in Java, communicated with FAM (also in Java) natively.

- **Using ACME with an Orb**

In order to achieve the integration described above, it was necessary to be able to speak to ACME through a CORBA-standard Orb. Since ACME, as distributed by CMU, is not CORBA-enabled, we produced a CORBA-enabled version of ACME,

which is available from the EDCS software repository that was set up at the University of Colorado for the second EDCS Demo Days.

- Translating ACME to ZD

Even with the orb in place, it was necessary to translate from ACME into ZD. This requires navigating over the ACME data structure. We used the Demeter technology from Northeastern University to facilitate this translation. We generalized this use of Demeter to form a system called *Persephone*, for which a provisional patent application has been filed.

- Displaying ZD

In order to be able to display ZD structures, we explored the use of several existing tools. For example, we can output ZD structures into LaTeX that is suitable for the IBM TechExplorer. We can also output ZD structures in a format suitable for an early version of MediaDoc's diagram generator.

3.4 Accomplishments vs. Objectives

The accomplishments identified above contributed to realizing the project objectives, as follows:

- Non-intrusive and cost-effective design rationale capture

The alternatives management capability in FAMILIAR provides a multi-layer capability to track design rationale at arbitrary levels of abstraction. The e-mail capture and batch input capabilities of FAM allow much of this information to be recorded without additional effort on the part of designers.

- Enhanced automated support for software/system understanding

FAMILIAR can respond to queries about survivability, compatibility, and coverage at many levels of abstraction, due to the reasoning capabilities of ZD.

- Enhanced ability to anticipate the impact of intended changes to a system

ZD provides a means for determining the impact of a new component. This capability has been demonstrated through the survivability example described earlier.

- Support for increasing formalization

FAMILIAR supports incremental formalism. The alternatives management system allows designers to begin describing systems informally, and to place the alternative designs in relation to one another. As distinguishing features are identified, this can be made incrementally more formal by describing features of the alternatives, and by constraining the features with rules.

Similarly, decomposition information can be added as needed to the design description. At this point, the structure can be translated into ACME, to allow the designer to add information about the relationships between parts of the architecture. Some of the FAM features are translated into formal information about the architecture components. The designer may then elaborate any of these features, as needed, to take advantage of the capabilities of ZD.

Throughout this lifecycle, from identification of alternatives to formal specification of architectural properties, FAMILIAR provides incremental support. Any increase in formalism results in increased feedback from the system. This stands in contrast to many formal systems, in which no benefit is delivered until the entire system has been expressed formally.

- Traceability of evolving design decisions

FAMILIAR provides a consistent way to represent detailed as well as high-level design decisions. Because successive "snapshots" of a design may be recorded as alternatives, FAMILIAR's support for comparing and contrasting alternatives provides insight into the evolution of the design.

- Decision support through disciplined evaluation of design alternatives

FAMILIAR provides a systematic way to specify the features of a design alternative, so that it can be evaluated in a disciplined way. Feedback obtained at any level of formalism can be recorded as additional information in the design description for future consumption.

- Increased integration and feedback between lifecycle phases

We have applied the FR description language, enhanced with the FAM schema of alternatives, features, and tradeoffs, to problems at various levels of abstraction, from high-level architecture to algorithm design. This shows that the FAMILIAR concepts can be used throughout the software evolution lifecycle, which we expect will facilitate iteration between phases.

3.5 Accomplishments vs. Evaluation Criteria

In the FAMILIAR proposal, we identified potentially quantifiable goals of accuracy and explanatory power. Although we learned a great deal from using FAMILIAR within our own efforts (see Section 4), we did not have the resources to run a pilot development project that would test these process characteristics.

4. APPROACH

The project was structured into three one-year phases, with a demonstration of capabilities at the end of each phase. The first year's work was driven largely by lessons

learned from previous work on KAPTUR and ZD. Years two and three were driven by feedback from the demonstrations and lessons learned during the preceding year's development.

4.1 Year One: Con-ops and Initial Prototype

We began the project by developing a Concept of Operations (con-ops) for the FAMILIAR system, and then evaluating alternative approaches to implementing it. One of our earliest decisions was that the alternatives management part of FAMILIAR should be developed anew rather than built on existing KAPTUR code. There were several reasons for this decision. The most recent version of KAPTUR was a proprietary system for which we did not have immediate source code access. While such access might have been negotiated, we knew that the changes to be made to the system (based on lessons learned) were substantial enough to put into question the value of a source code license. Developing the code anew would allow us to write the alternatives manager in Java, with all of that language's expected advantages for platform portability and web compatibility.

The initial prototype of the Formal Alternatives Manager (FAM) was specified in a document that translated the lessons learned from KAPTUR into functional recommendations for the new system. One of the most important recommendations was a more rigorous treatment of features. While in KAPTUR features were simply uninterpreted text phrases, in FAM the assignment of features to alternatives became subject to certain semantic consistency checks. Following review of the specification, the software was implemented in an ad hoc Java-based design. The class structure of this initial prototype was an artifact of decisions made with the predominant goal of demonstrating the tool at the first EDCS Demo Days. While not very maintainable, the prototype showed the functionality that we wanted to demonstrate.

Concurrent with this activity, work began on a version of ZD tailored to functional rationale capture. The starting point for this was a system called ZD 1, which had been developed by Beat Liver at the Swiss Federal Institute of Technology and Swiss Telecom. Liver's work was based on Dr. Allemang's 1990 dissertation at the Ohio State University. ZD 1 provides capabilities for software configuration, diagnosis, and survivability analysis. (It is described in Liver's dissertation, Thesis #1519, Ecole Polytechnique Federale de Lausanne, 1996.)

Formal calculations in ZD 1 were based on a type calculus that was not fully automated. Running the system relied on interactions with a human user who would provide answers to simple type-inclusion queries. For example: "Is x a member of the set $\{x, y, z\}$?" or "Is the set of integers from 0 to n a subset of the set of positive integers up to $n-1$ ". In the initial phase of the FAMILIAR project we extended ZD1 to overcome such limitations.

We considered redeveloping ZD in Java to permit tight integration with the alternatives manager. We rejected this approach because, as an inference system currently written in Lisp, ZD was not particularly amenable to implementation in an imperative programming

language. Moreover, we expected that in due time there would be adequate tools for integration between Lisp and Java-based systems, and it was not a good use of research funds to try to achieve this on our own. It was necessary, however, to port ZD from a custom Lisp dialect to Common Lisp, so that we could take advantage of off-the-shelf integration tools such as CORBA and achieve a reasonable level of platform independence.

Instead, makeshift integration mechanism using Unix pipes was put into place to demonstrate the interaction of the alternatives manager and ZD. While not a maintainable solution, and certainly not operating system independent, this form of integration allowed us to demonstrate the FAMILIAR concept in action, obtain feedback, and plan for a more principled design in the next phase.

4.2 Year Two: Design Rationalization and Usability Enhancements

Considerable effort was spent during the second year in developing a clean, maintainable design for FAM, the alternatives management part of FAMILIAR. The FAM user interface was still implemented using the Microline Component Toolkit (MCT) because we did not believe that Sun's Swing technology was sufficiently mature. However, we designed the new FAM so that MCT could be easily replaced by Swing when the time was right. In addition to paving the way for this transition, this approach allowed us to take advantage of the underlying Swing architecture, a form of the model-view-controller paradigm into which Sun had put considerable thought.

During this re-design of FAM, we made changes to the user interface in response to problems that beta-testers had voiced concerning the initial prototype. For example, the assignment and un-assignment of features to design alternatives, which had previously been achieved by special buttons, was changed to be done by copy/paste/cut/delete in the *Features of Design Alternative* window. (This itself was a compromise since we really wanted to use drag-and-drop, but it was not available in the MCT library nor in the standard Java AWT classes.) Error reporting was made more consistent, with two levels of information available (initial brief message and on-request detailed explanation).

Several major functional additions to FAM were implemented in this phase. Chief among these was a form of non-intrusive rationale capture based on e-mail. We came to this idea by analyzing our own patterns of design discussions, trying to explain our own resistance to using the initial prototype of FAM. We found that, next to face-to-face dialogue, e-mail was the team's de facto medium of choice for hammering out design decisions. We therefore implemented a capability to capture e-mails that were cc'd to a FAM address. The capture capability allows a FAM user, at her leisure, to map arbitrary portions of the e-mail messages to the design alternatives (nodes in the FAM database) that they refer to. Subsequent users then see these message snippets as annotations to the design alternatives.

The e-mail capture capability assumes that the FAM database is already populated. Use of the initial prototype showed that this too was a burden on software designers, especially when schedules are tight and the team is focused on finding the right solution, not on documenting its reasoning. The population process is especially burdensome when there are a lot of alternatives and features. While the user interface of FAM supports fully interactive tree editing to input this information, the process is not fast or easy enough to permit rapid input of large amounts of information.

In response to this scale-up problem, we implemented a batch-input function whereby FAM either creates or extends a database to include information contained in a tab-formatted ascii file. Tabs are used to specify the tree structure. This enables designers to use a highly efficient input mechanism, the text editor, to specify the alternatives and features being discussed by the design team.

Some viewers of the initial demonstration suggested that more semantics were needed for FAM features. For example, users wanted to be able to specify properties of feature combinations (good, bad, allowed, not allowed, etc.). Some viewers suggested that FAM could automatically generate design alternatives by blindly combining features, and then prune the resulting alternatives space either automatically, through rules, or interactively through user choice.

In response to these and other suggestions, we implemented an open-ended rule engine in FAM. The engine provides a simple interface that allows users to create their own constraint and transformation rules, which provide increased semantics to the FAM alternatives and features. The FAM rule engine is a port of the Hendrix system, which had been developed for AFRL under the Knowledge-Based Software Assistant (KBSA) program. To work with FAM, the Hendrix rule-creation and execution functions were ported from the CLIPS production system language to the Java Expert System Shell (Jess), a Java-compatible CLIPS lookalike. It is important to recognize that the FAM rule engine does not duplicate or replace any ZD functionality. It serves as an intermediate vehicle for enriched, but not fully formal, semantics. This is consistent with our goal to support the evolution from informal to formal semantics as a domain matures.

Concurrent with these enhancements to FAM, ZD was integrated with the architecture language ACME, and the supporting ACMESTudio tool. ACMESTudio provided a graphical interface for specifying component inter-connections within design alternatives as well as for displaying the results of a ZD analysis. In addition, the import/export between ZD and ACME provides a basis for future integration with other tools.

Finally, in preparation for the second EDCS Demo Days, we developed two realistic scenarios of military system evolution. The scenarios show how FAMILIAR reduces effort and improves the resulting decisions. The first scenario involves a surveillance and data fusion situation in which correct interpretation of data requires the introduction of a new processing capability. Since the host on which this capability resides is of a different type from the rest of the system, connectivity must be established between the systems. FAMILIAR uses functional representation to analyze a proposed solution, identify

disconnects, and suggest a fix. The second scenario illustrates the use of FAMILIAR in the design of a distributed C³I system. FAMILIAR generates alternative architectures for the system, given a range of available components. FAMILIAR then assists the designers in pruning the space of alternatives through tradeoff analysis, using information captured from ongoing e-mail discussions.

4.3 Year Three: Improved Con-Ops and Client-Server Architecture

The primary lesson that we obtained from feedback at the second EDCS Demo Days was that FAMILIAR was too passive. It did not clearly enough support an explicit process with well-defined payoffs. A typical reaction was something like, "It is very nice, but what exactly do I do with all this information?" Even worse was the comment, "So you put all this information in there, but what exactly does the tool itself do?"

In the final year of the project, we revisited the Concept of Operations and developed a more explicitly process-based user interface. The new con-ops and interface are based on the idea of *knowledge recycling*. In this approach, one solves a problem (such as a design or planning task) by looking for existing artifacts that can serve as a part or all of the solution, perhaps with some adaptation. Having solved the problem, one records it in the database so that it is now available for similar searches in the future.

The new user interface contains a diagram illustrating this process. Another diagram illustrates the sub-process of evaluating potential solutions by comparing and contrasting them (the heart of the rationale capture process). These diagrams, as well as the user's current position within the process, are visible to the user at all times. The user therefore knows where he has been and where he should be going.

A related critique of the FAM user interface was the explosion of many windows with inter-related information. In the first two versions of FAM (versions 0 and 1), each type of information was displayed, upon user request, in its own window. For example, each of the following types of information was displayed in its own window:

- Alternative designs for a given system type
- Features of a given system type
- Features of a particular alternative
- Components of a particular alternative
- Goals of a given system type
- Tradeoffs

Modification to the information in one window would automatically propagate to update the related information in all other windows. Nevertheless, the screen quickly became cluttered with numerous windows, and it became hard to keep track of what one was looking at, and how it all fit together.

In FAM 2, the most recent version of the alternatives manager part of FAMILIAR, the multiple windows are replaced with a single integrated display divided into frames with sliding borders. Each type of information has a fixed frame position, so that the user knows to look there for that type of information. Because the frames are not overlapping, all related information is visible at all times. However, the user can slide the borders to view more of a certain type of information and less of another.

The individual *Features of Alternative Design* displays were dispensed with. Instead, the global Alternatives and Features frame are now linked so that when a single alternative is selected, all features of that alternative are automatically highlighted. Conversely, when a single feature is selected, all alternatives possessing that feature are automatically highlighted. This linkage can be toggled on and off so that it does not interfere with the process of updating the database.

Another problem that we tried to address in this phase was the intrinsic intrusiveness of having to use a standalone tool, distinct from the tools already on the designer's desktop, in order to capture design rationale. Our solution to this problem was to make FAMILIAR a web application, accessible through a web browser at a URL that is specified at install time. We learned early in the project that applets were not an effective means of implementing complex functions, and that the trend in web applications was towards server-side processing. We therefore re-designed FAM to support a client-server architecture with a light-weight user interface. At the same time, we moved from the MCT widgets to Swing in order to keep in step with the latest Java developments.

The conversion to a client-server architecture was only partially successful, for reasons discussed below under Lessons Learned. We believe it remains the correct approach, but we realize now that an effective implementation requires a more thorough design change than originally thought.

5. LESSONS LEARNED

This section summarizes the main lessons that were learned during the course of the FAMILIAR project. Some of these were learned early and were addressed, to a greater or lesser extent, in the second and third years. Others point the way to future work.

5.1 Non-Intrusive Rationale Capture

The greatest obstacle to systematically capturing design rationale is the perception that it interferes with the momentum of the design process itself. We have tried to address this issue by separating the capture of raw data from the process of structuring and analyzing it. For example, FAMILIAR captures e-mail messages and places them in an archive, but allows the user to defer the process of annotating FAMILIAR data with portions of the message. This is an example of the overall rhythm of alternating production and reflection cycles that characterizes an effective software design process.

A more detailed analysis of the issues surrounding non-intrusive rationale capture was presented in the Final Report of a related project, the Phase I SBIR project entitled *Capturing Design Rationale in a Multi-Media Design Narrative*. There, we considered the cognitive and motor barriers to articulating rationale in the heat of software development, and we made several recommendations towards overcoming the barriers. The recommendations are fully consistent with FAMILIAR, and carry the simple idea of e-mail capture much farther. Although this has not been the main emphasis in the FAMILIAR project, we now recognize that a well-developed non-intrusive capture capability is a prerequisite for a tool like FAMILIAR to be widely used.

5.2 Integrated Display

Part of the value provided by FAMILIAR is its automated maintenance of inter-related information, and its support in visualizing the web of design information. Our naive assumption at the beginning of the project was that, in the absence of sophisticated display mechanisms, we could use basic window-icon-menu-pointer (WIMP) techniques, and the user would benefit simply from the presence of the information.

We discovered that this is not really the case. The multiplicity of windows creates a form of information overload in which the user quickly loses track of what information is being displayed, and where it can be found. We responded to this problem by changing to a single-window, multiple-frame user interface. This is less general than the multiple window approach because each type of information corresponds to a particular frame position. For example, there is only one Alternative Designs frame, one Features frame, and one Tradeoffs frame. (The user can get around this limitation by creating multiple web-browser windows, but we have not considered this as part of the recommended usage pattern.)

Although the new interface is less general, we believe this is an instance of the principle that Less is More. Viewers of the FAMILIAR demonstration at the third EDCS Demo Days responded favorably to the new interface. Those who had seen the multi-window interface concurred that the integrated display was much more comprehensible. Some of the limitations of the integrated display were overcome through interface dynamics (in effect, using time instead of screen space as a dimension). For example, the individual "Features of an Alternative Design" windows were replaced with automatic highlighting of the features of a selected alternative, within the global Features frame.

The integrated display clarifies the value provided by FAM as an information server. The Features and Tradeoffs frames, as well as the separate browser window in which URLs for selected alternatives are displayed, are all dynamically updated to reflect the currently selected alternatives. The user no longer has to request (through a button press) to see any of this information. As different alternatives are selected and the entire display changes, a picture of the data inter-relationships is conveyed with less prompting from the user than in previous versions.

Our long-term goal is to find novel visualization techniques, possibly using 3-D and animation, to provide even more highly integrated displays. This is an area for experimentation, which would be an important part of any follow-on work.

5.3 Pro-Active Guidance to the User

With the earlier versions of FAMILIAR, there was some confusion on the part demonstration viewers as to where the data in the tool came from, and how it was supposed to be used. Part of the confusion was an artifact of the demonstration setting, since we did not show the process of populating the tool with data. We did demonstrate the mechanics of entering data into the tool, when asked. However, the essence of the process is not data entry but rather the formulation of design alternatives through discussion, research, and reasoning. An integrated demonstration with a tool like Orbit or WinWin (two other EDCS projects) might have been an effective way to show this process, but limitations of time and funding prevented it from happening.

Demonstration viewers were therefore presented with an already populated database, and the demonstrations focused on the use of the recorded information. However, because the information was already recorded, and therefore the "answers" that FAMILIAR provided were already determined, the demonstrations lacked a convincing message of value provided.

Our analysis of this problem convinced us that it went deeper than demonstration mechanics. FAMILIAR, in its first two versions, resembled a spreadsheet tool. It was predominantly a passive system, in which the user entered information and specified links between certain information elements. The tool assisted the user by automatically maintaining values as determined by these links and by computing other values (such as tradeoffs). In addition, the ZD portion of the tool provided automated analysis of the information entered. There was, however, no explicit indication of the recommended use of the computed information (tradeoffs) or of the analysis results. A potential user who was already attuned to the implied methodology would see the value of the tool, but one who was not so attuned would not "get it."

This lesson was known to us from the days of KAPTUR. It did not find its way into the initial con-ops for FAMILIAR, perhaps because with so many other operational changes, we were not sure that the same problem would arise. When it became apparent that it did, we were able to bring to bear a past analysis of the problem for KAPTUR and propose an explicit process-directed user interface.

Under this interface, two levels of process diagrams serve as the vehicle for user commands. Each node in the diagram corresponds to an activity supported by FAMILIAR. The node acts as a button invoking the corresponding FAMILIAR function. Arrows between nodes represent the recommended flow of activities. Since the process diagrams are always displayed, they provide the user with a constant reminder of the

current activity, the activities that led up to the current one, and the recommended activities to follow, culminating in a problem-solving goal. When the goal is reached, the user has clearly accomplished something, and the role of the tool in helping to achieve it is apparent.

5.3.1 Pros and Cons of a Process-Based Interface

While we believe that the process-directed user interface is a major step forward in FAMILIAR's usability, there is a potential drawback to it, which we encountered during the third EDCS Demo Days. At least one viewer was concerned that any tool she acquire be tailorable to her organization's established processes. The explicit process diagram in the FAMILIAR display set off an alarm for this viewer. She asked us whether the process represented in the diagram was built into the tool or whether it could be modified. She expressed dissatisfaction with our answer that it was built into the tool.

To some extent, this was a problem of misunderstanding. The process represented in the FAMILIAR user interface is a very general pattern of knowledge recycling and systematic evaluation of alternatives in solving a problem. It is present, implicitly, in virtually any effective engineering process. It is made explicit in the FAMILIAR user interface because that is precisely the process that FAMILIAR aims to support. We do not believe that the "hard wiring" of this process into the user interface in any way limits the settings in which the tool can be used. However, its appearance may suggest this to potential users. Further enrichment of the interface, with tool tips and help text, could help mitigate this problem.

Another potential drawback of the explicit recommended process is that it may discourage novel and unexpected uses of the tool. The discovery of such uses is a well-known phenomenon in the evolution of successful tools, and we do not want to discourage it with FAMILIAR. However, at this early stage of the tool's existence, the problem of conveying the intended use must take precedence so that people will start to use the tool in the first place.

5.4 Need for DBMS Functionality

FAMILIAR may be viewed as an intelligent scratchpad for trading off design alternatives. While the principles of knowledge recycling apply to the work of a single designer as well as to that of teams, the need for tool support increases when there are multiple stakeholders with different goals, preferences, experience, and rationales. FAMILIAR was always envisioned as a multi-user tool, in which alternatives were proposed by different participants in the design process.

Multi-user access raises the issue of preventing concurrent conflicting updates to the database. In Version 1 of FAM, a modest level of support was provided by means of lock files. The locks ensured that any single *device type* (this is FR terminology for a type of

system, identifiable through its functional goals) could be modified by at most one user at a time. Because a device consists of components that are instances of other device types, the locks were required to extend to all devices in a *device group*. Device groups were an artifact that we created to refer to a hierarchy of devices.

We were never happy with the notion of a device group, and it disappeared from the user's view in Version 2. It was replaced with the idea of a single, corporate repository of design information. Unfortunately, this makes the locking capability even more crude than in Version 1. Locking the entire corporate memory for the duration of an entire user session is not an acceptable solution to providing data integrity.

What is clearly needed is a finer-grained locking capability. More generally, FAMILIAR would benefit from the kinds of multi-user, data integrity, and query support that an off-the-shelf Database Management System (DBMS) provides. Adding a DBMS to FAMILIAR should not be difficult, since the database currently consists simply of serialized Java objects, and Java itself provides an interface to ODBC, which many DBMSs support.

5.5 FAMILIAR as Corporate Memory

One of the challenges in describing FAMILIAR to potential users is to characterize the tool in terms that are already meaningful to them. This depends largely on the type of work the user does. FAMILIAR may alternatively be described as a decision support tool, a rationale capture tool, a knowledge management tool, a case-based reasoning tool, a domain modeling tool, a component repository, and an intelligent portal (see Section 5.6). FAMILIAR has some features of all of these, and yet it does not exactly fit the stereotype of any of these categories.

A common denominator in all of the descriptions is the idea of a corporate memory with pro-active support for problem solving. This is why we based the process-directed user interface of Version 2 on the idea of knowledge recycling. Tradeoffs and rationale capture play an essential but frequently unrecognized role in a corporate memory system. The motivation for "remembering" things in the system is to make use of them in the future. The purpose is to reconsider past decisions or to apply previous successes to new situations. In either case, it is important to understand why things were done as they were, and to evaluate potential alternatives. This relationship between corporate memory and rationale is the core idea of FAMILIAR, and we discovered that it takes a non-trivial effort to convey it to many potential users.

A related challenge is describing the intended and/or potential scope of FAMILIAR. As an EDCS tool, it was developed to support software evolution, as a way of evaluating alternative designs. Other kinds of software artifacts, besides designs, can also be represented as alternatives in the FAMILIAR database.

We have also demonstrated the application of FAMILIAR to logistics planning. When viewed as a decision-support or case-based reasoning tool, FAMILIAR can be seen to provide value to a wide range of problem-solving tasks. Alternatives are then to be viewed as alternative solutions to a given type of problem. To the extent that a problem can be decomposed into sub-goals that are functionally composed, even the FR component of FAMILIAR may be applicable.

This generality is, we have found, both a strength and a liability. It is a strength in that we can make a case for the use of the tool in many situations. It is a liability because people usually do not grasp generalities that they have not arrived at themselves. Selling FAMILIAR as a general-purpose corporate memory tool is probably not an effective communication strategy.

5.6 FAM as an Intelligent Portal

The recent emergence of portal technology for the web provides a handle for communicating the core idea of FAMILIAR. Viewing the web as a gigantic corporate memory (or as a vehicle for implementing an organizational memory) makes apparent the need for tools that focus attention on relevant information. Version 2 of FAM provides an explicit step in this direction by including a keyword-based search function. It can be argued, however, that the ability to compare and contrast items found on the web is as crucial as a search capability, since it provides a basis for filtering. Goal-oriented comparison immediately leads to consideration of tradeoffs and rationales, which is the heart of FAM.

In order to pitch FAMILIAR as an intelligent portal, we would have to populate it with a critical mass of web-based information. The information would have to be of a type that users expect to access via a portal. It is conceivable that in the future, a web-based market of software components could serve as such an application. Even today, web-based software archives contain enough alternatives to warrant FAMILIAR-type support in choosing components. (The obstacle in that case is the absence of economic incentive to implement the support.)

5.7 Design Trace vs. Domain Model

Closely related to the view of FAMILIAR as a corporate memory tool is the question of whether it supports system design or domain modeling. The underlying methodology draws a parallel between these two activities, and places FAMILIAR exactly in the middle. System design is a form of domain modeling, according to this methodology, where the domain consists of the space of possible designs. Conversely, domain modeling involves explicitly or implicitly envisioning alternative designs for a certain type of system. The difference, in principle, is one of scope.

In reality, system design and domain modeling are done by different people, at different times, funded by different sources, using different tools. The purpose of FAM is to bridge this gap. As such, one can consider the message to be "Let's bring the domain modeling back into system design." This is a call to place greater emphasis, time, and attention on the reflection cycle during software development, to ask on a continual basis, "What have we done here? How could we abstract it?"

The message can conversely be construed as "Let's bring the system design back into domain modeling." This is a call to include considerations about system structure (architecture and components) in the domain modeling process, which is sometimes limited to descriptive features and can thereby become divorced from the harsh realities of system development.

Positioning FAMILIAR mid-way between system design and domain modeling makes its role ambiguous, at least in the current state of the world. As in describing the tool as a form of corporate memory, the ambiguity is both an asset and a liability. It is an asset because it focuses attention on some important methodological truths (summarized in the messages above). It is a liability because, given the current way in which projects are organized, users do not clearly see how, when, or why to use the tool. Specifically, are they to use it during design to thrash out alternatives within a specific project, or during a domain modeling activity in which the products of multiple projects are considered as input? If the answer is "both," how are the project-specific traces of evolving alternative designs supposed to find their way into a cross-project organizational memory?

These issues were addressed in detail in the Final Report of a related project, a Phase I SBIR entitled *Models for Interoperable Rationale, Inference, and Alternative Designs* (MIRIAD). That report identified ways to extend FAMILIAR to support the transition from single-project to multiple-project mode and back.

5.8 Relationship with WinWin

Positioning FAMILIAR in relation to WinWin remained an issue throughout the project. Both tools are billed as rationale capture systems supporting the systematic evaluation of alternatives. In fact, the tools are quite different in terms of the functions they support. A consensus developed over the life of EDCS that they complemented each other. For example:

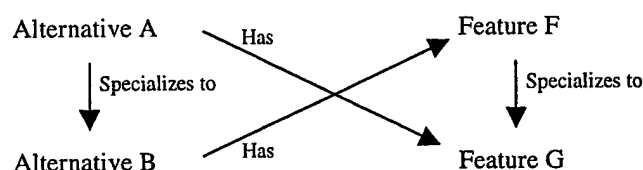
- FAMILIAR has a more semantically structured way of describing alternatives
- WinWin provides explicit support for multiple concurrent users
- FAMILIAR organizes information primarily in terms of product attributes
- WinWin organizes information around issues arising in a project

Where the consensus did not quite gel was in the optimal way to integrate the tools. Both projects agreed upon a straw-man mapping between WinWin and FAMILIAR data types.

Although it was expected that with experimentation, the mapping would probably change, the agreement was considered a good start. That the integration never occurred was an unfortunate function of priorities within each project.

5.9 Need for Rules

One of the first steps taken to move FAM beyond KAPTUR was to implement rules governing the assignment of features to alternatives. These rules constituted a subsumption check to the effect that a more specific feature could not be assigned to a more general alternative. In other words, the following pattern is not permitted to occur:



There are two degenerate cases of the check: A single feature cannot be explicitly assigned to a parent alternative and to one of that alternative's descendants. A single alternative cannot be assigned both an ancestor feature and one of that feature's descendants. Together, these rules ensure that descending the Alternatives Tree represents specialization. Equivalently, they ensure that the links in the Alternatives tree represent an "is-a" relation.

The subsumption rules were hard-coded into the Java implementation of FAM. Later, we implemented a similar set of checks for components of alternatives. There is an analogous rule to ensure that lower-level features are at least as good or at least as bad with respect to any goal as a higher-level (ancestor) feature.

All of these rules, especially the last, were called into question by at least some of the viewers of our demonstrations. Some viewers suggested additional types of rules, such as inferences about feature combinations, pruning of infeasible alternatives, and alternative computations of tradeoffs. These considerations led us to conclude that an open-ended means of specifying and implementing rules in FAM was needed. We accomplished this in the rule engine, which is an adaptation of the Hendrix tool developed under the KBSA program for AFRL.

5.10 Feature Combinations

At the demonstration during the first EDCS Demo Days, we received feedback from viewers that the tool needed more support for reasoning about feature combinations. Specifically, they wanted to be able to assign a "goal satisfaction" value not to an individual feature but to a combination of features. The tradeoffs of an alternative would

then take account of these values, factoring in the various combinations of features that an alternative possesses.

This request was communicated to us not as something desirable but as something essential for a methodologically sound tool. The point was made that an individual feature's contribution to (or detraction from) a particular goal is frequently meaningless. It must be considered in the context of the other features with which it co-exists.

We did not implement this function by the end of the project. However, by implementing the rule-engine we took a step towards supporting it. This was one of several potential enhancements to the decision support methodology (see Section 5.10, Tradeoff Calculations, for another). Since we expected others to arise, we thought it better to implement an extensible mechanism for specifying the methodology than to implement specific changes in the Java code.

The rule engine, implemented in Version 1.1 of FAM, does not achieve the requested capability. With some simple modification, however, it could. We envision providing the FAM user with the ability to assert facts about the database. A simple user interface for this process would have to be provided. Once the facts are asserted, they can be used as conditions that trigger the firing of rules.

For example, facts asserting the goal-satisfaction values of certain feature combinations could be used in rules that compute tradeoffs. The FAM rule engine does not currently provide a high-level user interface to specify such rules, which involve arithmetic processing, but it could be extended to do so. (The current high-level user interface supports the specification only of structural pattern detection and transformation rules.)

5.11 Tradeoff Calculations

FAM computes a goal-satisfaction value for each alternative by averaging over all of the alternative's features. As observed in Section 5.9, this approach assumes that feature interactions do not have an impact on the realization of a goal. Clearly, this is invalid as a blanket assumption.

Another weakness of the current tradeoff algorithm is that, after a value has been computed for each goal, the overall "goodness" of an alternative is computed by averaging over all goals, factoring in each goal's priority. In real-world decision situations, goals cannot accurately be given absolute priorities or weights. An alternative approach, known as the Analytic Hierarchy Process (AHP), is based on the idea that the evaluation of alternatives inevitably involves compromise. Therefore, a more realistic way to express priorities is to describe the *relative importance* of one goal to another, in a pair-wise fashion. Pair-wise ranking addresses the question, "If you must compromise on one of these two goals, which would it be, and how severe would the compromise be?" The AHP algorithm is a way of aggregating the pair-wise precedence relations in order to produce a composite "goodness" value for a proposed solution.

Replacement of the current algorithm with AHP would not be difficult. In fact, the weighted-average currently used was intended simply as a placeholder to illustrate the use of the tool. We assumed that a more sensitive algorithm could be found in the decision support literature and plugged in where the averaging algorithm is currently used.

5.12 Need for a Lightweight User Interface

We found that FAM was too large to run realistically as an applet. Running it in a browser required the Sun JDK plug-in, and even so it took a long time to load and, sometimes, exceeded available memory. Running it as a Java application also involved a significant wait while the classes loaded.

A rationale capture tool is already at a disadvantage with respect to user acceptance because the process it encourages appears to interfere with the progress of software development. An inordinately long load time for the tool is enough to ensure that it will not be used. This experience led us to conclude that FAMILIAR required a lightweight user interface that would load rapidly, preferably within a browser so that it did not appear as a separate tool.

The idea of a lightweight user interface also seemed compatible with a client-server architecture, which in turn would support integration with a DBMS than a better monolithic architecture. As we discovered, a lightweight user interface was not as immediate a consequence of a client-server architecture as we might have expected. This issue is discussed in Section 5.13.

5.13 Java Fragility

Working in Java provided many benefits, such as the ability to implement window-based interfaces rapidly using built-in language capabilities, and the ability to work with objects while not worrying about garbage collection. We learned, as many developers did during the same period, that applets do not really deliver on their promise. They take too long to download, and they stress browser performance and memory allocations to the point of exceptions or even crashing.

We developed Versions 0 and 1 of FAM to run as either an applet or an application. Running as an application is undesirable from the point of view of integration with existing desktop environments. Running as an applet proved impossible without using Sun's JDK browser plug-in, and only barely feasible with the plug-in.

We found that this was still true with the intended "lightweight" user interface of FAM Version 2. Because the user interface employed several Swing classes (such as JTree), the total volume of the applet, in terms of number of bytes to be downloaded, was not at

all "light." Using the JDK plug-in helps alleviate this problem because a local copy of Swing can be used, but this limits platform universality, the very benefit that running as an applet was intended to provide in the first place.

Another architectural problem prevented the Version 2 user interface from being as lightweight as we had expected. The problem results from using 1) Swing in the user interface, and 2) Remote Method Invocation (RMI) to communicate with the server. (Presumably, a similar problem would occur if RMI were replaced by CORBA calls).

The problem is as follows. The Swing architecture requires each user interface object (such as a tree) to have a corresponding *model* object, which is the logical representation of the information displayed in the widget. The goal of the client-server architecture is to move all significant processing to the server. It therefore makes sense to have the model objects reside in the server. This proved impossible to implement using RMI because of incompatibility between the interfaces that Swing models have to implement, and the interfaces that remote objects in RMI have to implement. In particular, exception handling proved impossible to implement with this design.

As a next-best approach, we tried keeping the model objects on the client side, but retaining the data objects (essentially, the individual data items that are linked together into trees, lists, etc.) on the server side. This too proved to be infeasible because of the frequency with which the model objects invoke methods of the data objects. The coupling between models and data is too tight to split over a remote interface. We therefore had to include widgets, models, and data in the client side, resulting in a user interface that was only slightly smaller than in previous versions. At the same time, because the server is now responsible for maintaining persistence and handling search requests, the data classes have to be replicated on the server side. This is not a desirable architecture.

We reached the conclusion that a truly lightweight user interface, and a maintainable client-server architecture, require dispensing with the applet entirely. This implies that all displays would be achieved through HTML, which is itself not a happy prospect. It might be possible to retain the existing tree processing logic by instantiating the model objects on the server side. Display update events would then be translated into dynamically generated HTML. Whether the HTML should be generated on the server side or through browser scripts would be a design issue.

The broader conclusion to be drawn from these considerations is that Java/web application technology is still in an immature state. While it is advancing rapidly, its use raises questions of stability and reliability for users whose satisfaction is our primary goal.

5.14 Impact of Using a Theorem Prover in ZD

Several technical advantages were gained by using a theorem prover to support the formal layer of ZD. For example:

The theorem prover permits a more compact and explicit representation of the information used in the Compatibility Constraint Satisfaction Problem (CCSP). This helps to clarify the algorithm, much of which concerns the propagation of information up and down the functional decomposition tree. When the underlying formalism is expressed in a theorem prover, propagation information can be expressed concisely in terms of the ordering of expressions in the theorem prover's language.

Much of the recursive Proviso Proving and Propagation (PPP) algorithm involves keeping track of the relationships between variables in various contexts. The contexts are defined by logical expressions in the language of the theorem prover. The algorithm was simplified by NQTHM's support for re-writing expressions given in one context into another context.

The 2-phase commit example was a particular challenge for the theorem prover. In this example, the designer uses FR to express the ways in which a certain state can be reached. The designer also specifies the conditions under which the state *must* be achieved in order to meet the requirements of the system. ZD then determines whether, taken together, all of the ways to achieve the state are sufficient to cover the desired conditions. This form of query can be difficult for a general-purpose theorem prover. ZD uses information from the functional representation to direct the theorem prover's search so that it can return an answer.

5.15 Interoperability of Formal Languages

In the FAMILIAR proposal, we suggested allowing components at different levels to be described in different logical languages. The motivation for this was that it would be common for the designers of components to express a component's function using the language most suited to that function.

In hindsight, this proposal seems ill advised. Many of the ZD algorithms, such as CCSP and PPP, rely in subtle ways on the details of the theorem prover. The dependencies include ways in which the algorithms manage information across levels. Rather than disrupt these dependencies, we achieved a similar goal in a different way. Providing a single but very expressive formalism such as that provided by NQTHM allows the designer to specify constraints that describe the terms used at each level. The introduction of *domains* allows whole sets of terms to be defined (and constrained) as a unit, thereby allowing for consistent sets of terminology to be used at each level.

6. FUTURE DIRECTIONS

Several areas of future work were identified in Section 5 along with the lessons learned that motivate them. In this section we summarize the most important of these for future reference.

The highest priority is to get FAMILIAR into a state in which it can be reliably deployed in real-world development situations. To do this, we must re-implement the client-server architecture of FAM with a truly lightweight user interface. As discussed in Section 5, this probably means foregoing the use of an applet entirely and relying instead on dynamic HTML. This is a non-trivial development exercise, involving some interesting architectural questions and potentially some issues about non-intrusive rationale capture. Nevertheless, given the experience gained in this project, we believe that it will be a relatively straightforward process.

Once the client-server architecture is stable, the next step towards equipping FAMILIAR for scale-up will be to integrate a DBMS. The one issue of substance in this task will be to streamline the FAM data model so that a user can easily "pan" in all directions over the corporate memory.

Some explanation of this challenge is in order. In FAM 2, we consider all alternatives to be arranged in a single global hierarchy that is available for search and browsing. A keyword-driven search function directs the user to the sub-trees most likely to contain the information he needs. However, we want to allow the user to move up, down, and across these sub-trees to view related information.

The "panning" process is not well supported in FAM 2 because of the baggage that has to be carried whenever new alternatives are placed in the user's view. This baggage includes components of the alternatives, which the user may also want to view. Therein lies the problem, because the components themselves may belong to very different locations in the global hierarchy. Therefore, moving up or down the alternatives tree involves more than just stepping across a single tree link. Efficiently handling this task will be the main challenge in providing a scaleable database.

A related task will be to integrate a robust search capability. Certain aspects of the search process are specific to FAMILIAR, such as the use of features to describe alternatives. Nevertheless, it would be foolish for us to try to replicate the power of existing search engines. We must find a way to use them in the context of FAMILIAR. (FAM 2 contains a very simple search function, which is intended as a placeholder to illustrate the process.)

A final task related to DBMS integration will be to provide fine-grained locking for multiple concurrent users.

The tasks just described support scale-up to real-world development environments. Another prerequisite to getting FAMILIAR used is to refine the non-intrusive knowledge capture capabilities. The e-mail capture function that we implemented in this project provides a good foundation. In order to make it truly attractive to developers, we must be able to test and refine it on an ongoing basis in a real development context. Although this may be considered a form of productization, it will provide us with deeper understanding of the usability issues involved in rationale capture.

A final task in support of true usability will be to support the transition between project-specific system design and organization domain modeling. The ambiguity in the current tool, discussed in Section 5.7, must be removed by providing explicit support for project design traces. The tool should support the transfer of project information into a cross-project corporate memory, and, conversely, importing information from the corporate memory for use in a specific project.

Our vision will be realized when it is standard operating procedure for developers to access the corporate memory and immerse themselves in multi-media presentations of relevant best practice. They will use FAMILIAR to analyze the suitability of these solutions to the current problem, and to adapt and compose them as appropriate. At each step of the process, they will use FAMILIAR to capture both formal and informal rationale, thereby weaving their experience into the ongoing design memory of the organization and contributing to its collective competencies.

LIST OF ACRONYMS

3-D	Three Dimensional
AFRL	Air Force Research Laboratory
AHP	Analytic Hierarchy Process
AWT	Abstract Window Toolkit
CCSP	Compatibility Constraint Satisfaction Program
CLIPS	C Language Integrated Production System
CMU	Carnegie Mellon University
ConOps	Concept of Operations
CORBA	Common Object Request Broker Architecture
DARPA	Defense Advanced Research Projects Agency
DBMS	Database Management System
EDCS	Evolutionary Design of Complex Software
FAM	Formal Alternatives Manager
FAMILIAR	Formal Alternatives Management Integrating Logical Inference and Rationale
FR	Functional Representation
HENDRIX	Help Evaluating New Designs with Rules Interactively Extendible
HTML	Hypertext Markup Language
JDK	Java Development Kit
JESS	Java Expert System Shell
KAPTUR	Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationale
KBSA	Knowledge-Based Software Assistant
MCT	Microline Component Toolkit
MIRIAD	Models for Interoperable Rationale, Inference, and Alternative Designs
ODBC	Object Database Connectivity
ORB	Object Request Broker
PI	Principal Investigator
PPP	Proviso Propagation and Proving
RC	Rationale Capture
RMI	Remote Method Invocation
URL	Universal Resource Locator
Y2K	Year Two Thousand
ZD	Zweck Darstellung

DISTRIBUTION LIST

addresses	number of copies
MS. DEBORAH A. CERINO AFRL/IFTD 525 BROOKS ROAD ROME NY 13441-4505	20
KNOWLEDGE EVOLUTION, INC. 1050 17TH STREET, NW, SUITE 520 WASHINGTON DC 20036	5
AFRL/IFOIL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	1
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ATTN: NAN PFRIMMER IIT RESEARCH INSTITUTE 201 MILL ST. ROME, NY 13440	1
AFIT ACADEMIC LIBRARY AFIT/LDR, 2950 P. STREET AREA B, BLDG 642 WRIGHT-PATTERSON AFB OH 45433-7765	1
AFRL/HESC-JDC 2698 G STREET, BLDG 190 WRIGHT-PATTERSON AFB OH 45433-7604	1

ATTN: SMDC IM PL
US ARMY SPACE & MISSILE DEF CMD
P.O. BOX 1500
HUNTSVILLE AL 35807-3801

1

COMMANDER, CODE 4TL0000
TECHNICAL LIBRARY, NAWC-WD
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6100

1

CDR, US ARMY AVIATION & MISSILE CMD
REDSTONE SCIENTIFIC INFORMATION CTR
ATTN: AMSAM-RD-OB-R, (DOCUMENTS)
REDSTONE ARSENAL AL 35898-5000

2

REPORT LIBRARY
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

1

ATTN: D'BORAH HART
AVIATION BRANCH SVC 122.10
FOB10A, RM 931
800 INDEPENDENCE AVE, SW
WASHINGTON DC 20591

1

AFIWC/MSY
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

1

ATTN: KAROLA M. YOURISON
SOFTWARE ENGINEERING INSTITUTE
4500 FIFTH AVENUE
PITTSBURGH PA 15213

1

USAF/AIR FORCE RESEARCH LABORATORY
AFRL/VSOSA(LIBRARY-BLDG 1103)
5 WRIGHT DRIVE
HANSCOM AFB MA 01731-3004

1

ATTN: EILEEN LADUKE/D460
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

1

OUSDO(P)/DTSA/DUTD
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

1

AFRL/IFT
525 BROOKS ROAD
ROME, NY 13441-4505

1

AFRL/IFTM
525 BROOKS ROAD
ROME, NY 13441-4505

1

CENTRIC ENGINEERING SYSTEM, INC.
624 EAST EVELYN AVENUE
SUNNYVALE, CA 94086-6488

1

FLUENT INCORPORATED
500 DAVIS STREET, SUITE 600
EVANSTON, IL 60201

1

THE MACNEAL-SCHWENDLER CORPORATION
815 COLORADO BOULEVARD
LOS ANGELES, CA 90041-1777

1

MOLECULAR SIMULATIONS, INC.
9865 SCRANTON ROAD
SAN DIEGO, CA 92121-3752

1

CENTRIC ENGINEERING SYSTEM, INC.
624 EAST EVELYN AVENUE
SUNNYVALE, CA 94086-6488

1